Schulze and Ranked-Pairs Voting are Fixed-Parameter Tractable to Bribe, Manipulate, and Control*

Lane A. Hemaspaandra, Rahman Lavaee, and Curtis Menton
Department of Computer Science
University of Rochester
Rochester, NY 14627, USA

October 25, 2012; revised December 17, 2012

Abstract

Schulze and ranked-pairs elections have received attention recently, with the former having quickly become the most widely used Condorcet method. For many cases these systems have been proven resistant to bribery, control, and manipulation, with ranked pairs being particularly praised for being NP-hard for all three of those. Nonetheless, the present paper shows that with respect to the number of candidates, both Schulze and ranked-pairs elections are fixed-parameter tractable to bribe, control, and manipulate: we obtain uniform, polynomial-time algorithms whose degree does not depend on the number of candidates.

1 Introduction

Schulze voting [Sch11], though relatively recently proposed, has quickly been rather widely adopted. Designed in part to well-handle candidate cloning, its users include the Wikimedia foundation, the Pirate Party in a dozen countries, Debian, KDE, the Free Software Foundation Europe, and dozens of other organizations, and Wikipedia says that "currently the Schulze method is the most widespread Condorcet method" [Wik12].

Although the winner-choosing process in Schulze voting is a bit complicated to describe, involving minima and maxima and comparisons of paths in the so-called weighted majority graph, finding the winner in a Schulze election nonetheless is polynomial-time computable. However, Parkes and Xia [PX12], followed by Menton and Singh [MS12], showed that for Schulze elections bribery is NP-hard, 15 of the 22 benchmark control attacks are NP-hard, and the complexity of manipulation is an open question (except it is in P if there is at most one manipulator).

*Supported in part by grants NSF-CCF-0915792 and NSF-CCF-1101479. Also appears as URCS-TR-2012-982.

Parkes and Xia also note that, by the work of [PX12,XZP$^+$09,Xia12], the ranked-pairs election system, which is not widely popular but like Schulze has a polynomial-time winner-determination problem and like Schulze is based on the weighted majority graph, is resistant to (basically, NP-hard with respect to) bribery, control (of each of the control types they study in their paper), and manipulation. Based on that discovery of ranked pairs being more broadly resistant to attacks than Schulze, and the fact that Schulze itself "is in wide use," and the fact that there is "broad axiomatic support for both Schulze and ranked pairs," Parkes and Xia quite reasonably conclude that "there seems to be good support to adopt ranked pairs in practical applications."

However, in this paper we show that the resistances to attacks of Schulze and ranked pairs are both quite fragile.

For each of the bribery/control/manipulation cases studied by Parkes and Xia, and Menton and Singh, for which they did not already prove Schulze voting to be in P (that is, they either proved the case NP-hard or left it as an open research issue), we prove that Schulze voting is fixed-parameter tractable (with respect to the number of candidates). Fixed-parameter tractable (see [Nie06]) means there is an algorithm for the problem whose running time is $f(j)I^{O(1)}$, where j is the number of candidates and I is the input's size. This of course implies that for each fixed number of candidates, the problems are in polynomial time, but it says much more; it implies that there is a global bound on the degree of the polynomial running time, regardless of what the fixed number of candidates is.

That result might lead one to even more strongly suggest the adoption of ranked pairs as an attractive alternative to Schulze. However, although for ranked pairs Parkes and Xia proved all the types of bribery, control, and manipulation they studied to be NP-hard, we show that every one of those cases is fixed-parameter tractable (with respect to the number of candidates) for ranked pairs. So even ranked pairs does not offer a safe haven from fixed-parameter tractability.

2 Presentation of Key Idea

Our proofs are of interest in their own right, because they face a very specific challenge, which at first might not even seem possible to handle. We now describe in relatively high-level terms what that challenge is and how we handle it.

Before we start that explanation, we need to present the definition of Schulze voting. Voters will always vote by linear orders over the candidates (in doing that, we adopt the complete, tie-free ordering case of Schulze used in both of the papers most related to this one [PX12,MS12]). Given the input set of candidates and the set of votes over them (as linear orders), the weighted majority graph (WMG) is the graph that for each ordered pair of candidates c and d, $c \neq d$, has an edge from c to d having weight equal to the number of voters who prefer c to d minus the number of voters who prefer d to c. Clearly, either all WMG edges have even weight or all WMG edges have odd weight, and the weight of the edge from c to d is negative one times the weight of the edge from d to c. The "strength" of a directed path between two nodes in the WMG is the minimum weight of all the edges along

that path (the strength can be negative). The Schulze election system is that candidate c is a winner exactly if for each candidate d it holds that there is some simple path from c to d whose strength is at least as great as that of every simple path from d to c. A lovely result is that the set of winners, under this definition, is always nonempty [Sch11]. Here is a small Schulze-election example from [PX12] over the set of candidates {*4*,*3*,*2*,*1*}. Although the votes are not specified here, we can build such a profile of votes to realize the WMG of figure 1 using McGarvey's method [McG53]. Candidate *4* is the sole Schulze winner,

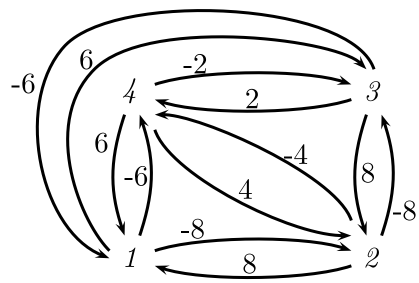

Figure 1: WMG for the election examples

strictly beating each other candidate in best-path strength. For each other candidate i, *4* has (in this example) a path to i of strength 6, but i's strongest path to *4* has strength 2.

One of the most powerful tools to use in building algorithms establishing fixed-parameter tractability is a result due to Lenstra, showing that the integer linear programming feasibility problem is in P if the number of variables is fixed [Len83]. Lenstra's result, based on the geometry of numbers, is very deep and so breathtakingly strong that intuition whispers it should not even be true; yet it is.

Now, if within a polynomial-sized integer linear program with a number of variables that was bounded by some function of the number of candidates we could capture our bribery/manipulation/control challenges and the action of the election system, we would be home free. Indeed, this has been done, for control, for such systems as plurality, veto, Borda, Dodgson, and others (see the discussion on page 338 of [FHH11]). However, Schulze and ranked-pair elections have such extremely demanding definitions that they seem well beyond such an approach, and we have not been able to make that approach work. So we have a challenge.

Fortunately, the literature provides a way to hope to approach even systems that are too hard to directly wedge (together with the manipulative action) into an integer linear program feasibility problem. That approach is to define some sort of structure associated with subcases of behavior/outcomes of an election system, such that for each fixed number of candidates the number of such structures is bounded as a function of the number of candidates (independent of the number of voters), yet such that for each such structure we can wedge into an integer linear programming feasibility problem the question of whether the given action can be made to succeed in the system in a way that is consistent with that

structure. If that can be done, then we just loop over all such structures (for the given number of candidates), and for each of them build and run the appropriate integer linear programming feasibility problem.

This has not been done often, but it has been done by Faliszewski et al. [FHHR09], with respect to some control problems, for the election system known as Copeland voting. And the structure they used is what they called a Copeland Output Table, which is a collection of bits associated with the outcomes of the pairwise majority contests between the candidates.

Unfortunately, such output tables don't seem to have enough information to support the case of Schulze or ranked pairs. The natural structure that would allow us to tackle our systems is the one the systems are based on, namely, the WMG, and looping over all of those would allow us within the loop to easily write/run an appropriate integer linear programming feasibility problem to check the given case. However, and this is extremely unfortunate, that falls apart because the number of WMGs is *not* bounded as a function of the number of candidates; the number also grows as a function of the number of voters. The impossibility of looping over WMGs leaves us still faced with the challenge of how to tackle our problems.

The central proof contribution of this paper is to show that the needle described above can be threaded—and to thread it—for Schulze and ranked-pair elections. In particular, we need to, for each of those election systems, find a structure that on one hand is rich enough that for each structure instance we can within an integer linear programming feasibility problem check whether the given manipulative action can lead to success in a way consistent with the case of which the particular instance of the structure is speaking. Yet on the other hand, the structure must be so restrictive that the number of such structures is bounded purely as a function of the number of candidates (independent of the number of voters). In brief, we need to find, if one exists, a "sweet spot" that meets both these competing needs.

We achieve this with structures we call Schulze winner-set certification frameworks (SWCFs) and ranked pairs winner-set certification frameworks (RPWCFs). A Schulze winner-set certification framework contains a "pattern" for how we can prove that a given set of candidates is the winner set of a Schulze election. To do that, the structure for each winner a specifies, for each other candidate b, a "strong path" γ_{ab} from a to b in the WMG (recall that victory in Schulze elections are based on having strong paths), and then to establish that the other candidate b has no stronger path back to a, for every simple path from b back to our candidate a, the structure identifies a "weak link" (a directed edge on that path) that will keep the path from being too strong; to be more specific, we mean an edge on that path in the WMG such that its weight is less than or equal to that of every edge in our allegedly quite strong path γ_{ab}. (Now, keep in mind, at the time we are looping through the structure, we will not even know how strong each link is, as the manipulation/bribery/control will not yet even have happened; rather, the structure is specifying a particular pattern of victory, and the integer linear programming feasibility problem will have to check whether the given type/amount of manipulation/bribery/control can bring to life that victory pattern.) And the structure for each candidate a it claims is not a winner will specify what rival b eliminates that candidate from the winner set and then outlines

a pattern for a proof that that is the case, in particular giving a "strong path" from b to a and for each simple path from a to b our structure specifies a "weak link," i.e., an edge on that path from a to b whose weight in the WMG we hope will be *strictly* less than the weight of all edges in the selected strong path from b to a; if all our hopes of this sort turn out to be true (and that is what the integer linear program will be testing, for each of our certification framework's structures), this proves that b eliminates a. Crucially, the number of structures (in that Schulze winner-set certification framework), though large, is bounded as a function of the number of candidates. The certification framework, however, does not itself have its hands on the weights of the WMG, and so the paths and edges it specifies are all given in terms of the self-loop-free graph, on nodes named $1, 2, \ldots, \|C\|$, that between each pair of distinct nodes has edges in both directions. (Since the candidate names are irrelevant in Schulze voting, we can change to those canonical names, so that our Schulze structures are always in terms of those names.)

Crucially, the number of structures (in that Schulze winner-set certification framework), though large, is bounded as a function of the number of candidates. Yet, also crucially, this approach provides enough structure to allow a polynomial-sized integer linear programming feasibility problem to do the "rest" of the work, namely, to see whether by a given type of attack we can bring to life the proof framework that a given instance of the structure sets out, as to who the winners/nonwinners are in the Schulze election and why.

For ranked pairs, the entire approach is just as described above, except the certification framework we use is completely different than that used for Schulze. Ranked pairs is a method that is defined in highly sequential terms, through successive rounds some of which add a relationship between two candidates, and so our certification framework will be making extensive guesses about what happens in each round (and about a number of other things). But again, we will ensure that the number of such certification structures is bounded as a function of the number of candidates (independent of the number of voters), yet each structure will give enough information that the rest of the work can be done by an integer linear programming feasibility problem. Our notion of a ranked pairs winner-set certification framework will be given in detail in Section 5.1.2.

3 Definitions

Schulze elections were defined in the previous section. We now define the quite different system known as ranked pairs, due to Tideman (see [Tid06]). Ranked pairs is also defined in terms of the weighted majority graph (WMG). The winner is defined by a sequential process. One chooses an edge in the WMG of greatest weight, say from a to b, and fixes in the eventual output that a must beat b. It then removes the edges between a and b from the WMG. It then iterates this process, except if the greatest remaining edge is one between two candidates who are already ordered (directly, or transitively) by earlier fixings of output ordering, then we discard the pair of edges between those candidates. We continue until we

have completely fixed a linear order.[1]

For clarity we give an example. We again consider the election with candidates $\{4,3,2,1\}$ and votes such that Figure 1 is the WMG. We will break order-of-consideration ties (due to tied edge weights) between $\{a, b\}$ and $\{c, d\}$ in favor of which pair has the lexicographically larger larger candidate, and if they tie in that, on which has the lexicographically larger smaller candidate. Thus we handle the edges in the following order: $3 \rightarrow 2$, $2 \rightarrow 1$, $4 \rightarrow 1$, $1 \rightarrow 3$, $4 \rightarrow 2$, $3 \rightarrow 4$. The output ordering will be set by those ($3 \succ 2$, $2 \succ 1$, etc.), except with $1 \rightarrow 3$ discarded due to transitivity. So under ranked pairs, 3 is the sole winner.

As mentioned earlier, our elections are specified by a set of candidates and voters (each vote is a tie-free linear ordering of the candidates). The standard (also called "nonsuccinct") approach to the votes is that each comes in separately. In the succinct approach (which is meaningful only for systems, such as Schulze and ranked pairs, that don't care about voters' names), each tie-free linear ordering that is cast by at least one voter comes with, as a binary integer, the number of voters that voted that way. We will in our problems speak of making a candidate p a winner or precluding p from being a winner, and this is known as the nonunique-winner model. If one changes "a winner" into "the one and only winner," that is what is known as the unique-winner model.

The bribery problem [FHH09] for an election system $\mathcal{E}$ takes as input an election specified by candidate set C, a vote collection V, a bribery limit k, and a distinguished candidate $p \in C$. The constructive (respectively, destructive) bribery problem asks whether there is a way of selecting and changing the votes of at most k voters (not all need to be changed to the same vote, though doing so is legal) such that p is (respectively, is not) a winner of the resulting election under $\mathcal{E}$. In the literature, this is called the unweighted, unpriced bribery problem.

The manipulation problem [BTT89,CSL07] for an election system $\mathcal{E}$ takes as input an

[1] There are two different types of ties that must be handled. One is when we get to a case when we are considering an edge, and we don't discard it, and the candidates tie (the edges between them are both 0); here, we break ties using some simple ordering among the candidates. By simple, we mean feasible; there is a P-time machine which given the candidates, outputs a linear ordering of them that is the ordering to use when breaking ties of this sort. The second type of tie is when there is a tie as to what is the largest edge remaining in the WMG. In ties of that sort, we use a simple ordering among unordered pairs of candidates to decide which pair having a highest-weight edge still left is the one to next consider. Again, by simple we mean feasible, analogously to the first case.

An at first seemingly tempting alternate approach would be to require as part of the input the two types of tie-breaking orders discussed above. But that is highly unattractive, since that would require changing the definitions of long-defined problems (manipulation, control, bribery), in order to add that extra input part. In truth, the tie-breaking is being made, by us and the earlier papers, to be a part of ranked pairs; and so it should be a feature or setting that is part of one's version of ranked pairs, and should not be built in by hacking the notions of manipulative actions. So to us, if one wants to speak about ranked pairs, one to be clear and complete must also specify the two feasible tie-breaking functions that are needed to completely define the system. However, our main results for ranked pairs, which are fixed-parameter tractability results, will all hold for all feasible tie-breaking functions (such as, for example, breaking ties between two candidates in favor of the lexicographically larger; and breaking ties between two candidate pairs in favor of the pair with the larger larger member and when the larger members are the same in both breaking the tie by who has the larger smaller member; the suggestion to use the candidate-vs.-candidate ordering to induce an ordering on the pairs was made by the creator of ranked pairs, Tideman, in [Tid06]).

election specified by candidate set C, nonmanipulative voter set V (each voting by a tie-free linear order), and manipulative voter set W (each starting as a blank slate), and a distinguished candidate $p \in C$. The constructive (respectively, destructive) manipulation problem asks whether there is a way of setting the votes of the manipulative voters such that p is (respectively, is not) a winner of the resulting election under $\mathcal{E}$. In the literature, this is called the unweighted coalitional manipulation problem.

As benchmarks, over time, 11 "standard" types of control questions have emerged, each with a constructive and destructive version. Four of the 11 are each of adding/deleting (at most k, with k part of the input) candidates/voters. A fifth is so-called unlimited adding candidates. The remaining 6 are partition of candidates, run-off partition of candidates, and partition of voters, each in both the model where first-round ties promote and in the model where first-round ties eliminate. Detailed, formal definitions can be found in [FHHR09]. However, briefly put, all these problems have as their input an election, (C, V), and a distinguished candidate $p \in C$. Constructive (destructive) control by deleting voters—for a given election system, of course—also has a nonnegative integer k in the input and asks whether there is a subset of V of cardinality at most k such that with that subset removed p is (is not) a winner. Control by adding voters is analogous, except the input is the election, k, and a set W of voters who can be added (but at most k can be added). Deleting candidates and adding candidates are analogous to the voter cases, with a k as part of the input, and the only twist is that in destructive control by deleting candidates, it is forbidden to delete p. Unlimited adding candidates is the same except there is no limit k. Constructive (destructive) partition of voters, in the ties-promote model, asks whether there is a way of partitioning the voters into two groups so that if all winners under the election system of each of those first-round elections compete in a final election under the same election system in which all voters vote (with their votes masked down to the remaining candidates), p is (is not) a winner. In its ties-eliminate variant, only unique-winners of a first round election move forward. The run-off partition of candidates types are analogous, except in the first round it is the candidates that are partitioned and all voters vote in each of those subelections. Partition of candidates has just one side of the partition participating in the first-round election, while the others get a bye to the final round.

A problem is said to belong to the class FPT (is said to be fixed-parameter tractable, see [Nie06]) with respect to a parameter if there is an algorithm for the problem whose running time is $f(j)I^{O(1)}$, where j is the input's value of that parameter, f is a computable function, and I is the input's size. Crucially, note that this means that although the algorithm for larger values of j can have a bigger multiplicative constant, the degree of the polynomial running time is uniformly bounded from above—there is some single integer k such that regardless of the fixed j the algorithm for that parameter bound runs in time $O(n^k)$. Our parameter will almost always be the most natural one—the number of candidates. In Section 7, we will have cases where our parameter is a tuple of features of the input rather than a single feature, i.e., is a "combined" parameter (see [Bet10, Chapter 9]).

4 Related Work

The computational complexity of manipulation, bribery, and control for Schulze voting and ranked pairs has been studied previously by Parks and Xia, Xia et al., Xia, and Menton and Singh [PX12,XZP^{+}09,Xia12,MS12]. Briefly summarized, these papers establish (see the Table in [PX12]) that for Schulze elections constructive and destructive bribery, constructive and destructive control by adding and deleting voters, and constructive control by adding candidates are all NP-complete, and that ranked pairs has not just all these hardnesses but also has NP-completeness results for constructive and destructive manipulation, for destructive control by adding candidates, and for constructive and destructive control by deleting candidates. Menton and Singh (see Table 1 of [MS12]) studied all 11 types of constructive and all 11 types of destructive control for Schulze voting, even those not considered in the earlier work, and showed that for 15 cases NP-completeness holds and for 7 cases P algorithms exist. All the results in the three papers involving Xia are in the unique-winner model. Ranked pairs is resolute (has exactly one winner), as Parkes-Xia frame it, and we follow their framing. And so the unique-winner and the nonunique-winner models are in effect the same for ranked pairs. Schulze is not resolute, but although their results on that are in the unique-winner model, they comment that their results all also hold in the nonunique-winner model. Menton and Singh use the nonunique-winner model as their basic model, as do we in the present paper; it is to us more attractive in not requiring a tie-breaking that, especially in symmetric cases, is often arbitrary and can change the flavor of the system. However, our main FPT results are proven by a loop approach over ILPFPs, and it is clear that a straightforward adjustment to these will also handle the unique-winner cases. The key difference between our work and the mentioned work is that our work is in general looking at the complexity of these problems when parameterized by the number of candidates, and for this we give FPT algorithms, and the earlier papers looked at unbounded numbers of candidates and obtained both P and NP-completeness results. Our contribution is that for all their NP-complete cases, we show membership in FPT.

As to technique, the closest precursors of this paper are two papers by Faliszewski et al. [FHHR09,FHH11]. Those, like us, use a loop over ILPFPs. The main differences between that work and ours is that (a) they deal with control, and we also are concerned with bribery and manipulation, and (b) as explained in detail in Section 2, their type of loop-over structure isn't flexible enough for our cases, and the natural structure for us to loop over generates a number of objects not bounded in the number of candidates, and so we in this paper find a middle ground that allows the loop to be over a bounded-in-$\|C\|$ number of objects yet provides enough information in the objects so as to allow the ILPFPs to complete the checking of whether success is possible. For a different type of attack known as "swap bribery" and a different election system, Dorn and Schlotter [DS12] have recently employed what in effect is a loop over ILPFPs, and they mention in passing without details that that swap bribery approach should apply to ranked pairs.[2]

[2]They separately mention in passing and without definitions or details that their approach should apply

Looking at things from an even broader perspective, this work is part of a line that looks at the complexity of elections in the context of bounds on the number of candidates, a study that for example has been pursued famously by Conitzer, Sandholm, and Lang [CSL07] regarding at what candidate numbers things jump from P to NP-complete. The particular focus on FPT algorithms, and maintaining a uniform degree bound over all values of bounds on the number of candidates, is part of the important field of parameterized complexity (see [Nie06], and see [BBCN12] for a survey on this approach for elections).

5 Results by Looping over Frameworks

We now present our results that are established by our looping-over-frameworks idea. We will handle in separate sections bribery, manipulation, and control, showing how to achieve FPT results for each. Within the bribery section, we will first prove the bribery result for Schulze elections, so that the reader gets quickly to seeing how the proof goes without having to have first seen how the approach works for ranked pairs. We then will give our ranked pairs winner-set certification framework, and will note how to convert our proof into a proof for that case also. Then in the manipulation and control sections, we will state and prove together the Schulze elections case and the ranked pairs case.

5.1 Bribery Results and Specification of Ranked Pairs Winner-Set Certification Framework

We shall now state and prove the bribery case for Schulze.

---

to manipulation and some unspecified variants of control. If one takes as implied there the combination of their ranked-pairs aside and their other attacks-aside (and they in their paper don't explicitly assert that), then that without-details assertion pair, combined, overlaps some of our ranked-pairs results regarding control (though they do not specify which control types they are speaking of). However, in contrast, we here actually provide a certification framework handling the ranked pairs election system. We suspect their results even if looking at the combination of their asides don't overlap our main work on manipulation, since they speak of "weighted and unweighted manipulation," but by that undefined-there use they seem to mean a weaker notion of manipulation than that used in the present paper, namely, we are looking at coalitional manipulation, but they seem to be referring to noncoalitional manipulation. (The reason we say this is that for the weighted case of coalitional manipulation, the natural ILPFPs one would generate have neither their number of variables nor their number of constraints "bounded in the number of candidates independently of the number of voters." Beyond this, for ranked pairs we will prove, as Theorem 7.3, that weighted coalitional manipulation is NP-complete for each fixed number of candidates starting at five; this easily implies that unless $P = NP$ no FPT algorithm can exist. There we will discuss how far toward such algorithms one can seem to get within the ILPFP approach and its extensions, namely, by looking at the special case of bounded weights and even of bounded weight-set-cardinality. That section notes that we can handle even the coalitional case for bounded weights, and indeed even for unbounded weights but bounded manipulator-weight-set-cardinality, and that latter case itself is a generalization of weighted noncoalitional manipulation.)

5.1.1 Bribery Result for Schulze

Theorem 5.1 *For Schulze elections, bribery is in FPT (is fixed-parameter tractable) with respect to the number of candidates, in both the succinct and nonsuccinct input models, for both constructive and destructive bribery, in both the nonunique-winner model and the unique-winner model.*

Proof. Our FPT algorithm works as follows. It gets as its input an instance of the bribery problem, and so gets the candidates (with a distinguished candidate noted), the votes (or for the succinct version, a list of which types of votes occur at least once, along with the multiplicities of each), and the limit k on how many voters can be bribed. Let j be the number of candidates in the input instance. To mesh with the naming scheme within our SWCFs, we immediately rename all the candidates (including within the votes) to be $1, \ldots, j$, with the distinguished candidate becoming candidate 1. Now, the top-level programming loop of the algorithm is the following:

Algorithm 1 Top level loop for bribery

Start
for each j-SWCF, (call it) K **do**
  if candidate 1 is a unique winner according to K and K is an internally consistent, well-formed j-SWCF **then**
    (1) build an ILPFP that checks whether there is a way of bribing at most k of the voters such that K's winner-set certification framework is realized by that bribe
    (2) run that ILPFP and if it can be satisfied then halt and accept (note: the satisfying settings will even let us output the precise bribe that succeeds)
  end if
end for
declare that the given goal cannot be reached by k bribes
End

All that remains is to specify the ILPFPs that we build inside the loop, for each given j-SWCF K. Suppose we are doing that for some particular K. We do it as follows.

There are $j!$ possible votes over j candidates; let us in any natural, computationally simple to handle way number them from 1 through $j!$. We call the ith of these the ith "vote type." We will have constants n_i, $1 \leq i \leq j!$, denoting how many voters start with vote type n_i. Our ILPFP will have integer variables (which we will ensure are nonnegative) $m_{i,\ell}$, $1 \leq i \leq j!$, $1 \leq \ell \leq j!$. $m_{i,\ell}$ is the number of voters who start with vote type i but are bribed to instead cast vote type ℓ. (Having $m_{i,i} \neq 0$ is pointless but allowed, as is having simultaneously $m_{i,\ell} > 0$ and $m_{\ell,i} > 0$.) So the number of ILPFP variables is $(j!)^2$, which is large but is bounded with respect to j, so Lenstra's algorithm can be used to deliver an FPT performance overall.

Also, not as direct parts of the ILPFP but as tools to help us build it, we define two boolean predicates, *Bigger*(a, b, c, d) and *StrictlyBigger*(a, b, c, d), where the arguments each

$$\left(\sum_{\substack{\{i \mid 1 \leq i \leq j! \text{ and} \\ \text{in votes of type } i \text{ it} \\ \text{holds that } a \text{ is pre-} \\ \text{ferred to } b\}}} \left(n_i - \left(\sum_{1 \leq \ell' \leq j!} m_{i,\ell'} \right) + \left(\sum_{1 \leq i' \leq j!} m_{i',i} \right) \right) \right) - \left(\sum_{\substack{\{i \mid 1 \leq i \leq j! \text{ and} \\ \text{in votes of type } i \text{ it} \\ \text{holds that } b \text{ is pre-} \\ \text{ferred to } a\}}} \left(n_i - \left(\sum_{1 \leq \ell' \leq j!} m_{i,\ell'} \right) + \left(\sum_{1 \leq i' \leq j!} m_{i',i} \right) \right) \right)$$

$$\geq 1 + \left(\sum_{\substack{\{i \mid 1 \leq i \leq j! \text{ and} \\ \text{in votes of type } i \text{ it} \\ \text{holds that } c \text{ is pre-} \\ \text{ferred to } d\}}} \left(n_i - \left(\sum_{1 \leq \ell' \leq j!} m_{i,\ell'} \right) + \left(\sum_{1 \leq i' \leq j!} m_{i',i} \right) \right) \right) - \left(\sum_{\substack{\{i \mid 1 \leq i \leq j! \text{ and} \\ \text{in votes of type } i \text{ it} \\ \text{holds that } d \text{ is pre-} \\ \text{ferred to } c\}}} \left(n_i - \left(\sum_{1 \leq \ell' \leq j!} m_{i,\ell'} \right) + \left(\sum_{1 \leq i' \leq j!} m_{i',i} \right) \right) \right).$$

Figure 2: Constraint enforcing that, after the bribes happen, $\boldsymbol{D(a,b) > D(c,d)}$.

vary over $1, \ldots, j$. Let us use $D(a,b)$ to indicate the weight of the WMG edge (after our manipulative actions) that points from a to b. Recall, from Section 2, that what K does is specify (for a very large number of such quadruples, though that number actually is bounded as a function of j though we do not need that since our number of variables is already bounded as a function of j) that $D(a,b) \geq D(c,d)$ or that $D(a,b) > D(c,d)$, i.e., that in the WMG, a certain edge is greater than or equal to another edge in weight, or is strictly greater in weight. For each such specified relation that explicitly appears in K, set to true that bit in the appropriate predicate, and leave all the other bits set to false. We of course will have to enforce these specifications through our constraints.

Now, we can specify all the constraints of our ILPFP. There will be three types of constraints. The first are the housekeeping constraints to make sure that the number of bribes and the $m_{i,\ell}$s are all reasonable. Our constraints of this sort are: For each $1 \leq i, \ell \leq j!$, we have a constraint $m_{i,\ell} \geq 0$. For each $1 \leq i \leq j!$ we have a constraint $n_i \geq \sum_{1 \leq z \leq \ell} m_{i,z}$. And we have the constraint $k \geq \sum_{1 \leq i', \ell' \leq j!} m_{i',\ell'}$.

The second type of constraint is those to enforce the bits set to "true" in *StrictlyBigger*. For each such bit, we will generate one constraint: for a bit that is saying that $D(a,b) > D(c,d)$, we will enforce that with the constraint shown in Figure 2. All that bulky-looking constraint says is that after all the gains and losses due to bribing happen, the number of voters who prefer a to b minus the number who prefer b to a is strictly larger than the number of voters who prefer c to d minus the number who prefer d to c.

The third type of constraint is those to enforce the bits set to "true" in *Bigger*. For each such bit, we will generate one constraint: for a bit that is saying that $D(a,b) \geq D(c,d)$, we will enforce that with precisely the constraint shown in Figure 2, except with the "1+" removed from the second line.

That completes our statement of the ILPFP, which indeed captures what it seeks to capture. And using Lenstra's algorithm for each of our ILPFPs, the overall loop over the ILPFPs has the desired running time. (Although for each fixed j the multiplicative constant is very large, the degree of the polynomial, which is uniform over all j, isn't

terrible; Lenstra's algorithm uses just a linear number of arithmetic operations on linear-sized integers [Nie06]. Still, even within the good news that we have placed the problem within FPT, there is the bad news that the multiplicative constant is so large that this FPT algorithm does not provide an algorithm for practical use.) To be clear, as what is a constant and what is a variable is a bit subtle here, let us say a bit more about the use of Lenstra here. What we in effect are using is that (for each fixed number of candidates it holds that) for each one of the (large but bounded as a function number of the number of candidates) ILPFPs generated in our loop, viewing that one ILPFP as an object whose running time for solution is being evaluated (asymptotically, as the number of voters increases without bound), that that running time by Lenstra is polynomial with, indeed, a uniform upper bound on the degree, independent of the number of candidates and independent of which of our ILPFPs for the given problem we are speaking of. Note that each such ILPFP object in effect has as *its* set of variables (regarding the asymptotics of its running time) the *constants* of the ILPFP; and a big part of what our looping algorithm does is to set those constants based on the votes in the election. ❑

To change the above proof from the constructive to the destructive case, and/or from the nonunique-winner case to the unique-winner case, we in the main loop simply create ILPFPs for only those SWCFs whose set of who wins and loses reflects a sought outcome (e.g., for the destructive case in the unique-winner model, that would be having the distinguished candidate not be a unique winner).

The same approach directly applies to ranked pairs. However, it requires some extra work to define the winner-set certification framework for ranked pairs as we defined it for Schulze. Generally, the applicability of our approach relies only on existence of such frameworks that precisely specify the winner-set using a bounded number of constraints, rather than on any specific internal structure of those.

5.1.2 Specification of Ranked Pairs Winner-Set Certification Framework

We now describe the analogous winner-set certification framework that makes our approach work for ranked pairs.

Basically, an instance of that framework will be a story that tells us what happens at each stage of the iterative process that defines ranked pairs. We could actually tell this story without fixing up front which of each pair of candidates is preferred to the other by a majority of the voters, or whether they exactly tie in that. That would improve our multiplicative constant, but we are not focused on that constant, so to make things particularly simple to describe, we are here just going to toss into our framework a fixing of all such pairwise outcomes in the WMG. As in the Schulze case, we will have changed all the names of the candidates to be 1 through $\|C\|$ (and will have remapped our tie-breaking function in the same way). So, one part of our framework is, for each (unordered) pair of candidates (a, b), a claim as to which one of these holds: the WMG edge from a to b is strictly positive, the edge from b to a is strictly positive, or both edges are 0. (We do not include any claim about the precise value of those edge weights; that would create a

framework whose number of instances, for a fixed number of candidates, grew with the number of voters—anathema to us.) And an instance of the framework then goes step by step through the process the ranked-pair algorithm goes through, but in a somewhat ghostly way in terms of what it specifies. For each step of the process, it makes a claim as to what pair of candidates is considered next, and a claim as to whether that pair of candidates will be skipped permanently due to it having been already set (due to transitivity) by earlier actions of our flow through ranked pairs (that isn't an on-the-fly thing in that the instance itself has all its earlier claims and so we can even make sure to only loop over instances of the framework that are internally consistent regarding this), and if it is not skipped, a claim about which of the two outcomes happens (which is placed above the other in our ranked-pair outcome; again, we can read this from those choice-of-3-possibilities settings we did up front, plus the feasible tie-breaking if needed). So that is the story the framework provides, and a given instance of the framework will (if properly formed) set an ordering over all the candidates. As before, the algorithms will loop over instances of these frameworks, doing so over only instances that have the desired outcome (e.g., "p is a unique winner") and that aren't obviously internally inconsistent. (For our partition by voter cases, there is a double-loop over such frameworks, to handle both subelections.)

As for the Schulze case, we will use the ILPFPs to see if the given kind of control can create a case where the given framework can be made to hold. All the housekeeping work in the ILPFPs as to tallying how the votes are bribed/controlled/manipulated is still needed here (so the variable sets are the same as the ones for Schulze). But note, crucially, that we now must enforce not things about paths, but rather we must enforce that the framework's guesses about whether the edge from a to b is negative, positive, or 0 after the bribery/control/manipulation are all correct (this is very natural to enforce with constraints, within the ILPFP framing), *and must also enforce that the framework's claim about which candidate pair is considered next is what would actually happen under the votes that emerged from the bribery/control/manipulation.* But that latter claim, for each step in the story, can be checked by appropriate, carefully built constraints, written with close attention paid to the tiebreaking rule among pairs. These constraints will be pretty much our favorite sort of constraints—seeing whether a WMG edge is greater than or equal to another, or seeing whether it is strictly greater than another. This will be made clearest by an example. Suppose our candidates are named 1, 2, 3, 4. And suppose the tiebreaking order on unordered pairs is $\{4,3\} > \{4,2\} > \{4,1\} > \{3,2\} > \{3,1\} > \{2,1\}$, and on candidates is $4 > 3 > 2 > 1$. Suppose the RPWCF says the first pair to be compared is $\{4,1\}$ and that the outcome is $4 \succ 1$. Let $D(a,b)$ be defined as before. To check that $4 \succ 1$ is the right outcome, since $4 > 1$ in the tiebreaking function we need to check that $D(4,1) \geq D(1,4)$ (if $1 \succ 4$ in the tiebreaking order, we'd check that $D(4,1) \geq 1 + D(1,4)$); we can read this right off the 3-way-claim as to how 1 and 4 compare in their head-to-head contest, which itself we'll enforce in constraints. And as to the claim that $(4,1)$ was the first pair to be compared, in light of the tiebreaking order, that can be enforced using 10 constraints: 6 saying that our pair ties or beats those below us in the tiebreaking ordering ($D(4,1) \geq D(a,b)$ for the (a,b) values $(3,2)$, $(2,3)$, $(3,1)$, $(1,3)$, $(2,1)$, $(1,2)$), and 4 saying

that our pair strictly beats those above us in the tiebreaking ordering ($D(4,1) \geq D(a,b)$ for the (a,b) values $(4,3)$, $(3,4)$, $(4,2)$, $(2,4)$). We could cut those $6+4$ constraints to $3+2$ if we wish, by using the value of the 3-way-claim for each of those 5 other pairs. Note that all these comparisons are about post-bribe/manipulation/control vote numbers—things we do know how to easily put into an ILPFP constraint, basically, by appropriate summations. Moving on, if the framework says the next unordered pair after $\{4,1\}$ to be considered is $\{3,1\}$ and that the outcome is $1 \succ 3$, we of course will not need to enforce any comparisons with $D(4,1)$ or $D(1,4)$; we will generate and put into the ILPFP just the needed/appropriate comparisons. If the framework after that says the third pair to consider is $\{3,4\}$ but it also says that (due to $4 \succ 1 \succ 3$ already being set) the pair $\{3,4\}$ gets skipped, we under our RPWCF framework still must generate the constraints to check that $\{3,4\}$ truly under our votes as they now are *did* deserve to come up next (we mention in passing that we could skip that check as long as we adjust our ILPFP to not check anything regarding pairs that are already related, even transitively, under the $\succ$'s so far—a slightly different approach than ours but also quite fine), but the generated comparisons to check that won't do comparisons against things the existing order so far ($4 \succ 1 \succ 3$) takes out of play. This completes our description of our RPWCF notion.

5.1.3 Bribery Result for Ranked Pairs

Having specified the ranked pairs winner-set certification framework, the bribery case for ranked pairs directly follows that for Schulze.

Theorem 5.2 *For ranked pairs (with any feasible tie-breaking function), bribery is in FPT (is fixed-parameter tractable) with respect to the number of candidates, in both the succinct and nonsuccinct input models, for both constructive and destructive bribery, in both the nonunique-winner model and the unique-winner model.*

Proof. We use the same programming loop of Algorithm 1 to loop over j-RPWCFs with j being the number of candidates. The variables of the ILPFPs will be the same as those for Schulze. As for the constraints of the ILPFPs, all the housekeeping constraints for bribery remain intact. Additionally an RPWCF, as described in Section 5.1.2, guesses about every edge being positive, negative, or zero. We can easily handle these possibilities with the constraints $D(a,b) \geq 1$, $D(b,a) \geq 1$, and $D(a,b) = 0$ respectively. The other constraints enforced by an RPWCF are those between pairs of edges and they can be successfully captured by the *StrictlyBigger* and *Bigger* predicates as defined in the proof of Theorem 5.1. Thus we can build an ILPFP encoding an RPWCF and the constraints of the bribery problem in the same way we did for Schulze elections, and bribery is in FPT for ranked pairs as well. ❑

In the rest of the paper, we will jointly state (and prove) the results for Schulze and ranked pairs.

5.2 Manipulation Results

In this section, we show that our looping-over-frameworks approach can be used to obtain FPT algorithms for manipulation by embedding appropriate constraints into the ILPFPs enforcing the winner-set certification frameworks.

Theorem 5.3 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, manipulation is in FPT with respect to the number of candidates, in both the succinct and nonsuccinct input models, for both constructive and destructive manipulation, and in both the nonunique-winner model and the unique-winner model.*

Proof. Let us again assume without loss of generality that the candidates in the manipulation problem are $1, \ldots, j$ with the distinguished candidate being 1. The manipulation problem has as its input a set of nonmanipulative votes V (or for the succinct version, a list of which types of votes occur at least once, along with the multiplicities of each) and the set of manipulators W. The top-level programming loop will be as described in Algorithm 2 (with WCF being SWCF for Schulze and RPWCF for ranked pairs).

Algorithm 2 Top level loop for manipulation

Start
for each j-WCF, (call it) K **do**
  if candidate 1 (is/is not) a (unique/nonunique) winner according to K and K is an internally consistent, well-formed j-WCF **then**
    (1) build an ILPFP that checks whether there is an assignment to the votes of W such that K's winner-set certification framework is realized by the set of votes $V \cup W$

    (2) run that ILPFP and if it can be satisfied then halt and accept (note: the satisfying settings will even let us output the precise manipulation that succeeds)
  end if
end for
declare that the given goal cannot be reached with $\|W\|$ manipulators
End

This is the same loop as we had for bribery with only step 1 changed. So all we need to do is to specify the ILPFP that we build inside the loop, for each given j-WCF K.

Let us again number the vote types from 1 through $j!$. We will have constants n_i, $1 \leq i \leq j!$, denoting how many nonmanipulative voters cast a vote of type i. We will also have integer variables m_i, $1 \leq i \leq j!$, representing the number of manipulators selected to cast vote type i.

We have two basic types of constraints that we must implement in our ILPFP: Those formulating the WCF-enforcing predicates, and those enforcing the validity of the manipulative action. Recall from Section 5.1.1 that each predicate is a strict or non-strict inequality relation between weights of two edges in the WMG. (In ranked pairs there is an additional

set of constraints enforcing the weight of each edge to be zero, negative, or positive). So, if we are capable of expressing the weight of each edge as a linear expression in terms of our variables and constants, we can express the whole inequality as a linear constraint. Now in order to make more concise the representation of the constraints, let us introduce the shorthand notation $\mathit{pref}(a, b)$ as the set of vote types i, $1 \leq i \leq j!$, in which candidate a is preferred to candidate b.

Again, we denote by $D(a, b)$ the weight of the WMG edge from a to b after our manipulative action. We can express $D(a, b)$ as follows:

$$\sum_{i \in \mathit{pref}(a,b)} (n_i + m_i) - \sum_{i' \in \mathit{pref}(b,a)} (n_{i'} + m_{i'}).$$

Now every predicate of type $\mathit{StrictlyBigger}(a, b, c, d)$ can be implemented in our ILPFP as the following linear constraint:

$$\sum_{i \in \mathit{pref}(a,b)} (n_i + m_i) - \sum_{i' \in \mathit{pref}(b,a)} (n_{i'} + m_{i'}) \geq 1 + \sum_{i \in \mathit{pref}(c,d)} (n_i + m_i) - \sum_{i' \in \mathit{pref}(d,c)} (n_{i'} + m_{i'}).$$

The same constraint except with the "1+" removed from the right hand side expresses a predicate of type $\mathit{Bigger}(a, b, c, d)$.

In ranked pairs, the constraints on each of the three possibilities for an edge (explained in the proof of Theorem 5.2) will be implemented in terms of $D(a, b)$ as specified above.

In the second type of constraints, we have to enforce that the variables of the ILPFP form a valid manipulation. That is, they must sum up to the number of manipulators $\|W\|$.

$$\sum_{1 \leq i \leq j!} m_i = \|W\|.$$

Also, we need all of our variables to be nonnegative, thus we have $m_i \geq 0$ for every $1 \leq i \leq j!$.

The ILPFP we have described has encoded within it the problem of how to enforce a given WCF with a given set of manipulators. Thus the top-level algorithm specified above will solve the manipulation problem in uniform polynomial time for every fixed number of candidates, putting this problem in FPT. ❑

5.3 Control Results

In this section, we prove our results for control gained through our looping-over-frameworks approach. Our proofs for these case will closely follow our general looping-over-frameworks structure, and we will just have to appropriately build the constraints of our ILPFPs to handle the details of the control problems.

Theorem 5.4 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, control by adding voters is in FPT with respect to the number of candidates, in both the succinct and nonsuccinct input models, for both constructive and destructive control, and in both the nonunique-winner model and the unique-winner model.*

Proof. Again, let the candidates in the control problem be $1, \ldots, j$ with the distinguished candidate being 1. The control problem has as its input a set of initial votes V and a set of additional votes W (or for the succinct version, one list for each of these two sets describing which types of votes occur at least once in that set, along with the multiplicities of each), the later of which contains the votes to be added by the control action. Also there is a limit k on the number of votes to be added from W.

The top-level programming loop is as described in Algorithm 3 (with WCF being SWCF for Schulze and RPWCF for ranked pairs).

Algorithm 3 Top level loop for control by adding voters

Start
for each j-WCF, (call it) K **do**
  if candidate 1 (is/is not) a (unique/nonunique) winner according to K and K is an internally consistent, well-formed j-WCF **then**
    (1) build an ILPFP that checks whether there is a set of votes $W' \subseteq W$, with $\|W'\| \leq k$, such that K's winner-set certification framework is realized by the set of votes $V \cup W'$
    (2) run that ILPFP and if it can be satisfied then halt and accept (note: the satisfying settings will even let us output the precise added set that succeeds)
  end if
end for
declare that the given goal cannot be reached by adding at most k voters
End

All we have to do now is show how we build the ILPFP inside the loop. Again, as with manipulation, we will have two groups of constraints: those corresponding to the WCF-enforcing predicates and those corresponding to the structure of the control problem. For every i, $1 \leq i \leq j!$, we will have a variable v_i representing the number of votes of type i in W', a constant n_i representing the number of votes of type i in V, and a constant h_i representing the number of votes of type i in W.

As we described in the proof of Theorem 5.3, if we can represent $D(a, b)$ as a linear expression in terms of our constants and variables, we can implement all the WCF-enforcing constraints (those of the *StrictlyBigger* and *Bigger* predicates and those of the 3-way possibilities for ranked pairs) as linear constraints in our ILPFP. Therefore, using the shorthand notation $\mathit{pref}(a, b)$ as described in Section 5.2, we express $D(a, b)$ as follows:

$$\sum_{i \in \mathit{pref}(a,b)} (n_i + v_i) - \sum_{i' \in \mathit{pref}(b,a)} (n_{i'} + v_{i'}).$$

As for the constraints ensuring the validity of the control action, we first need to ensure that for every type of vote, the number of votes of that type in W' is bounded by the number in W. For every vote type i, $1 \leq i \leq j!$, the constraint $v_i \leq h_i$ will enforce this in our ILPFP. We also make the following constraint to enforce the adding bound:

$$\sum_{1 \leq i \leq j!} v_i \leq k.$$

Here again all of our variables have to be nonnegative, and thus we have the constraint $v_i \geq 0$ for every i, $1 \leq i \leq j!$.

This suffices to describe how we can build a WCF-enforcing ILPFP for the control by adding voters problem in a way appropriate for our looping-over-frameworks technique. Thus we have an algorithm that will run in polynomial time for every fixed parameter value, putting this problem in FPT. ❑

Theorem 5.5 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, control by deleting voters is in FPT with respect to the number of candidates, in both the succinct and nonsuccinct input models, for both constructive and destructive control, and in both the nonunique-winner model and the unique-winner model.*

Proof. The input for this problem is a control instance with a set of votes V over j candidates $1, \ldots, j$, with candidate 1 being the distinguished candidate, and with a bound k on the number of votes to be deleted. Algorithm 4 specifies the top-level loop.

Algorithm 4 Top level loop for control by deleting voters

Start
for each j-WCF, (call it) K **do**
  if candidate 1 (is/is not) a (unique/nonunique) winner according to K and K is an internally consistent, well-formed j-WCF **then**
    (1) build an ILPFP that checks whether there is a subset of the voters V', with $\|V'\| \leq k$, such that K's winner-set certification framework is realized by the set of votes $V - V'$
    (2) run that ILPFP and if it can be satisfied then halt and accept (note: the satisfying settings will even let us output the precise deleted set that succeeds)
  end if
end for
declare that the given goal cannot be reached by deleting at most k voters
End

The ILPFP we build inside the loop will include a constant n_i, $1 \leq i \leq j!$, representing how many votes of each vote type i are in the initial election. We will include in the ILPFP variables v_i, $1 \leq i \leq j!$, representing the number of votes of type i that are deleted. With this new variable, we will formulate $D(a, b)$ in the ILPFP as follows:

$$\sum_{i \in pref(a,b)} (n_i - v_i) - \sum_{i' \in pref(b,a)} (n_{i'} - v_{i'}).$$

With this we can easily build the *Bigger* and *StrictlyBigger* predicates we use to enforce the WCF, as well as the additional constraints necessary for a RPWCF.

Additionally, we need to include constraints that ensure that the control action chosen through the assignment to the variables is a legal one. We include a constraint $v_i \geq 0$ for every i, $1 \leq i \leq j!$, enforcing that we cannot delete a negative number of voters of any type. We include constraints $n_i \geq v_i$ for every i, $1 \leq i \leq j!$, enforcing that we cannot delete more voters of any type than there were in the first place. And finally we add a constraint $k \geq \sum_{1 \leq i \leq j!} v_i$, bounding the total number of deletions by the bound k.

These modifications to our basic approach embed the problem of deciding what subset of voters to delete to satisfy a given WCF into an ILPFP, and there will only be a constant number of ILPFPs to test for each number of candidates. Thus the algorithm specified by the outer loop above is uniformly polynomial for every fixed parameter value, putting this problem in FPT. ❑

Theorem 5.6 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, control by partition of voters is in FPT with respect to the number of candidates, in both the ties-eliminate and ties-promote models, in both the succinct and nonsuccinct input models, for both constructive and destructive control, and in both the nonunique-winner model and the unique-winner model.*

Proof. These cases can again be handled with our looping-over-frameworks approach, and the constraints and modifications necessary to properly implement the WCF-enforcing ILPFP are similar to the previous cases, though we have to loop over not just single WCFs but rather pairs of them, one for each part of the partition. As before we will assume the candidate names are $1, \ldots, j$ with 1 as the distinguished candidate. We specify the outer loop for these cases in Algorithm 5.

Now we must specify the new details of the partition-handling ILPFPs. For every i, $1 \leq i \leq j!$, we will include constants n_i representing how many votes of each vote type i are in the initial election and variables v_i representing how many votes of type i are placed in V_1.

Though we must use two WCFs, we can implement each as before, and in parallel in a single ILPFP. Let us use the predicates Bigger_1 and $\mathit{StrictlyBigger}_1$ (Bigger_2 and $\mathit{StrictlyBigger}_2$) to handle the constraints of WCF K_1 (K_2) over the weight of its WMG edges which we will denote by D_1 (D_2).

Bigger_1 and $\mathit{StrictlyBigger}_1$ can be formulated appropriately in terms of D_1, while Bigger_2 and $\mathit{StrictlyBigger}_2$ can be formulated in terms of D_2. The extra constraints for ranked pairs will be implemented in terms of D_1 and D_2 as appropriate as well. We will now show how to handle D_1 and D_2. $D_1(a, b)$ denotes how many voters in V_1 prefer candidate a to candidate b and we can formulate it as follows:

Algorithm 5 Top level loop for control by partition of voters

Start
for each j-WCF, (call it) K_1 **do**
  for each j-WCF, (call it) K_2 **do**
    if K_1 and K_2 are internally consistent, well-formed j-WCFs **then**
      (1) build an ILPFP that checks whether there is a partition of the voters V into V_1, V_2, such that K_1's winner-set certification framework is realized by the set of votes V_1 and K_2's winner-set certification framework is realized by the set of votes V_2
      (2) run that ILPFP and if it can be satisfied, and if candidate 1 (is/is not) a (unique/nonunique) winner in the election with voters V and the surviving candidates from K_1 and K_2 (according to the appropriate tie-handling rule), halt and accept (note: the satisfying settings will even let us output the precise partition that succeeds)
    end if
  end for
end for
declare that the given goal cannot be reached by any partition of voters
End

$$\sum_{i \in pref(a,b)} (v_i) - \sum_{i \in pref(b,a)} (v_i).$$

$D_2(a,b)$ denotes how many voters in V_2 prefer a to b, and we formulate it as follows:

$$\sum_{i \in pref(a,b)} (n_i - v_i) - \sum_{i \in pref(b,a)} (n_i - v_i).$$

Finally, we must add constraints to ensure the chosen partition is a legal one. We include a constraint $v_i \geq 0$ for every i, $1 \leq i \leq j!$, enforcing that a nonnegative number of voters of each type are chosen for the first partition. We include a constraint $n_i \geq v_i$ for every i, $1 \leq i \leq j!$, enforcing that we do not take more voters than exist of each type for the first partition.

These modifications to our basic approach embed the problem of deciding what partition of voters to use to satisfy a pair of WCFs into an ILPFP, and there will only be a constant number of ILPFPs to test for each number of candidates. Thus the algorithm specified by the outer loop above is uniformly polynomial for every fixed parameter value, putting this problem in FPT. ❑

6 Other Results

6.1 Candidate Control Parameterized on the Number of Candidates

Under our primary parameterization of interest, parameterizing on the number of candidates, and when considering manipulation, bribery, and voter control, we achieved FPT results through great effort using our looping-over-frameworks technique. In contrast, when considering candidate control problems parameterized on the number of candidates, we need not use such a powerful technique. Instead it is sufficient to brute-force search over all possible control solutions to see if any of them are successful. At every possible value for the parameter, there are only a constant number of possible solutions to any of the candidate control problems, and checking the success of the possible solution will only require a simple polynomial-time task. Thus the full procedures at every fixed parameter value will run in time of a large constant times a small uniform polynomial, putting the problems in FPT.

Theorem 6.1 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, control is in FPT with respect to the number of candidates, in both the succinct and nonsuccinct input models, for both constructive and destructive control, for all standard types of candidate control (adding/unlimited adding/deleting candidates, and, in both the ties-eliminate and ties-promote first-round promotion models, partition and runoff partition of candidates), in both the nonunique-winner model and the unique-winner model.*

Proof. In the case of adding candidates, at most all the $2^{\|D\|}$ possible subsets of the auxiliary candidate set need be considered. In the case of deleting candidates, we must consider at most the $2^{\|C\|-1}$ possible subsets of the candidate set to delete. In the case of runoff partition, there are $2^{\|C\|-1}$ distinct partitions, while in the partition case there are $2^{\|C\|}$. The difference between the partition cases is because the runoff partition the two parts are handled symmetrically, while they are not in the partition case. For any of these cases, we have to at most run three iterations of the voting system function, so all these cases will be in FPT for any voting system with a polynomial-time winner problem. ❑

6.2 Voter Control Parameterized on the Number of Voters

Although we feel that the number of candidates is the most natural parameterization for manipulative action problems, it is natural to ask about parameterizing on the number of voters. We do not exhaustively handle this case for all manipulative action problems, but we note that voter control problems parameterized on the number of voters can be shown to be in FPT through simple brute-force. Again, as in the case of candidate control problems parameterized on the number of candidates, we note that the number of possible solutions to these problems is bounded by a constant for each parameter value, and checking each solution is easily done in polynomial time, giving us an FPT algorithm for each of the voter control problems.

Theorem 6.2 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, control is in FPT with respect to the number of voters, in both the succinct and nonsuccinct input models, for both constructive and destructive control, for all standard types of voter control (adding/deleting voters, and, in both the ties-eliminate and ties-promote first-round promotion models, partition of voters), in both the nonunique-winner model and the unique-winner model.*

Proof. In the case of adding voters we will have to try at most all of the $2^{\|W\|}$ possible subsets of the auxiliary voter set. In the case of deleting voters we will have to consider at most all the $2^{\|V\|}$ subsets of the voter set. And in the partition of voters cases we will have to consider the $2^{\|V\|-1}$ distinct partitions of the voter set. In all of these cases the large exponential term of the complexity will be constant with fixed parameter values. And beyond that term, we will just have to perform a few iterations of the voting system's winner function along with a few other simple checks, putting all these cases in FPT for any voting system with a polynomial-time winner problem. ❑

6.3 WMG Edge Bound Parameterization

Let us return to considering the case of parameterization by number of candidates. What drove us to our approach of looping over the winner "frameworks" we defined, rather than just looping over all WMGs? It was the fact that even for fixed numbers of candidates, the number of WMGs blows up as the number of voters increases. We mention in passing, though, that if one, *in addition* to parameterizing on the number of candidates, requires that the absolute value of the edge weights in the WMG be bounded by some fixed constant independent of the number of voters, then for that particular special case, one could loop over all WMGs. Is this natural and important? We would tend to say "no," because assuming that all edges in the WMG have weights bounded by k is to assume that even as the number of voters grows, every single head-on-head contest between pairs of candidates is very evenly matched. That simply is not the case in most natural elections.

On the other hand, perhaps surprisingly, there is something theoretical to be gained from the strange approach just mentioned of considering elections in which all edges of the WMG turn out to have relatively low weights. In particular, we observe that in the NP-hardness-establishing reductions *to* Schulze control problems used by Menton and Singh [MS12], all edges in the WMG have absolute value at most 6 (and for some types of control, at most 4 or 2). That, along with that fact that all edges in the WMG have the same parity, gives us the following result.

Corollary 6.3 (Corollary to the proofs of [MS12].) *Even when restricted to elections having all pairwise contests so equal that each WMG edge has absolute value at most 2 (or alternatively that the total cardinality of the set of absolute values of WMG edges is at most 2; or alternatively that the total cardinality of the set of values of WMG edges is at most 3), Schulze elections are NP-complete (in the nonunique-winner model) for constructive*

control by deleting candidates. The same holds for constructive control by adding candidates, unlimited adding of candidates, partition of candidates in the ties-eliminate model, and run-off partition of candidates in the ties-eliminate model, except with the 2/2/3 above replaced by 4/3/5. The same holds for constructive control by partition of candidates in the ties-promote model and run-off partition of candidates in the ties-promote model, except with the 2/2/3 above replaced by 6/4/7.

6.4 Adding/Deleting Bound Parameterization

For those control problems having as part of their inputs a limit on how many candidates or voters can be added/deleted, it is natural to consider parameterizing on that limit. This parameterization has been studied in some voting systems and the relevant problems have often been found to be W[1]-hard or W[2]-hard, and thus very unlikely to be fixed-parameter tractable. For example, under this parameterization, Betzler and Uhlmann [BU09] showed, for what are known as Copeland$^\alpha$ elections, that constructive control by adding candidates and constructive control by deleting candidates are W[2]-complete, Liu and Zhu [LZ10] proved, for maximin elections, that constructive control by adding candidates is W[2]-hard, and Liu and Zhu also achieved W[1]-hardness results for the relevant voter control problems. For additional control results parameterized on the problem's internal addition/deletion limit, see Table 8 of [BBCN12].

We observe that, with respect to parameterizing on the internal addition/deletion bound, for Schulze elections, constructive control by adding voters and constructive control by deleting voters are both W[2]-hard. This can be seen through simple modifications of the NP-hardness proofs for these cases given by Menton and Singh [MS12]. Namely, we appropriately modify the proofs to reduce from the W[2]-complete problem hitting set instead of from exact cover by three-sets. Additionally, constructive control by adding candidates can be seen to be W[2]-hard for Schulze elections through a natural reduction from hitting set.

Theorem 6.4 *Constructive control by adding candidates parameterized on the adding bound is W[2]-hard for Schulze voting.*

Proof. We will reduce from the W[2]-hard problem hitting set.

Definition 6.5 (Hitting Set) *Given a set of elements U, a collection $\mathcal{F}$ of subsets of U, and a positive integer k, does there exist $H \subseteq U$, with $\|H\| \leq k$, such that for every $S \in \mathcal{F}$, we have $S \cap H \neq \emptyset$ (i.e. H hits every set in $\mathcal{F}$)?*

Given a hitting set instance $(U, \mathcal{F}, k)$ as described in the definition, we will construct a control instance (C, D, V, p, k) where C is the set of original candidates, D is the set of auxiliary candidates, V is the set of voters, p is the distinguished candidate, and k is the adding bound. The original candidate set C will contain the following:

- The distinguished candidate p.

- A candidate S for every $S \in \mathcal{F}$.

The auxiliary candidate set D will contain the following:

- A candidate u for every $u \in U$.

The voter set V will be as follows. We will not explicitly construct the entire voter set, but rather we will specify the weight of the WMG edges between the candidates and let the voter set be as constructed by McGarvey's method [McG53].

- For every $S \in \mathcal{F}$, $D(S, p) = 2$.
- For every $u \in U$, $D(p, u) = 2$.
- For every $u \in U$, and for every $S \in \mathcal{F}$ such that $u \in S$, $D(u, S) = 2$.
- All other WMG edges will be of weight 0.

The distinguished candidate will be p and the adding limit will be the same limit k as in the hitting set instance. This completes the specification of the reduction.

If we map from a positive hitting set instance, we will have a positive instance of the control problem. Let $H \subseteq U$, $\|H\| \leq k$, be a solution to the hitting set instance. We will show that the set of candidates D' corresponding to the elements from H will be a solution to the control instance. First, $\|D'\| \leq k$, so we are within the adding bound. Also, since the hitting set solution includes members of every set in $\mathcal{F}$, there will be a path from p to every candidate corresponding to those sets, as including a candidate $u \in U$ creates paths from p to every set S hit by u. So p will have paths to every other candidate just as strong as they have back to p, and so p will be a winner.

If we map to a successful control instance, we must have a positive hitting set instance. Succeeding in control will require us to create paths to every candidate that has a path to the distinguished candidate, so we must add a set of candidates that give us all the necessary paths. But we are limited to adding at most k of the candidates from the auxiliary set. It is then easy to see that a solution to our control problem, creating paths to every adversary candidate with a bounded number of additions, corresponds to a solution to the hitting set instance.

This proof is an FPT reduction. It clearly runs in polynomial time in the size of the entire input (exceeding the requirements for an FPT reduction), and the parameter in the mapped-to instance will always be bounded by (in fact, identical to) the parameter in the mapped-from instance. So this proof shows that constructive control by adding candidates parameterized on the adding bound is W[2]-hard for Schulze voting. ❑

7 Weighted Case

Finally, although in this paper we have focused on manipulation problems without weights, and on bribery problems without weights or prices, we mention in passing that (keep in mind we still are also parameterizing on number of candidates, and that when weights and prices are used in problems, they are typically taken to be nonnegative integers) if one parameterizes by also bounding the maximum voter weight (if there are weights) and the maximum voter price (if there are prices), our main theorems hold even in the context of weights and prices; that is because when weights and prices are bounded, one can clearly still carry out the approach we use.

In fact, we can go slightly further, though at the outer edge of things, doing so requires some surgery on our approach. Not just for the cases of bounded weights and prices, but even for the case where there is a bound on the *cardinality* of the set of weights (if there are weights) and there is a bound on the *cardinality* of the set of prices (if there are prices), our three main theorems (again, still parameterizing also on number of candidates) still hold. To give an example of how we can handle this, we now state and prove the case of bribery.

Theorem 7.1 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, weighted, priced, and weighted and priced bribery is in FPT with respect to the combined parameter "number of candidates" and "cardinality of the set of voter weights" and "cardinality of the set of voter prices" in both the succinct and nonsuccinct input models, for both constructive and destructive manipulation, and in both the nonunique-winner model and the unique-winner model.*

Proof. We will describe how to handle the weighted and priced case, and note that this is a generalization of either the weighted and unpriced case or the unweighted and priced case, so this algorithm will also work for either of these. The top-level loop will be as in the standard bribery proof in Section 5.1. We will specify the extensions to the ILPFP construction that are necessary to handle the weights and prices.

Let $1, \ldots, j$ be the candidates, let $w = \{w_1, \ldots, w_y\}$ be the bounded-cardinality set of weights of the voters, and let $p = \{p_1, \ldots, p_z\}$ be the bounded-cardinality set of prices. We will have a constant $n_i^{\alpha,\beta}$ denoting how many voters there are of type i with weight w_α and with price p_β for every i, $1 \leq i \leq j!$, α, $1 \leq \alpha \leq y$, and β, $1 \leq \beta \leq z$. We will have a variable $m_{i,\ell}^{\alpha,\beta}$, for every i, $1 \leq i \leq j!$, ℓ, $1 \leq \ell \leq j!$, α, $1 \leq \alpha \leq y$, and β, $1 \leq \beta \leq z$. $m_{i,\ell}^{\alpha,\beta}$ describes the number of voters with weight w_α and price p_β that are bribed from vote type i to vote type ℓ. Our total number of variables is larger than in the unweighted unpriced case, but it is still bounded in our parameters j, y, and z.

Now we will describe how to build the constraints in our ILPFP. As before, we can implement the predicates *Bigger* and *StrictlyBigger* and the additional constraints to handle ranked pairs in terms of $D(a, b)$, the weight of the edge from a to b in the WMG, so we need only specify how to formulate $D(a, b)$. We need to take into account both voters that are bribed away from preferring a over b and voters that are bribed towards preferring a over b. And we need to keep track of voters of every type, weight, and price. Still, what we

need is very much in line with our simpler implementation of this function in the standard bribery case. So our new formulation of $D(a,b)$ will be the following:

$$\sum_{1\leq\alpha\leq y}\sum_{1\leq\beta\leq z}\sum_{i\in pref(a,b)} w_\alpha\left(n_i^{\alpha,\beta}-\sum_{1\leq\ell\leq j!}\left(m_{i,\ell}^{\alpha,\beta}\right)+\sum_{1\leq\ell\leq j!}\left(m_{\ell,i}^{\alpha,\beta}\right)\right)$$
$$-\sum_{1\leq\alpha\leq y}\sum_{1\leq\beta\leq z}\sum_{i\in pref(b,a)} w_\alpha\left(n_i^{\alpha,\beta}-\sum_{1\leq\ell\leq j!}\left(m_{i,\ell}^{\alpha,\beta}\right)+\sum_{1\leq\ell\leq j!}\left(m_{\ell,i}^{\alpha,\beta}\right)\right).$$

Additionally we need to implement constraints to make sure that the bribery action we select is a legal one. Therefore, besides the natural constraints $m_{i,\ell}^{\alpha,\beta}\geq 0$, ensuring that we do not bribe a negative number of voters of a certain type, we have to enforce that we do not bribe more voters than exist of every type. Thus for every i, $1\leq i\leq j!$, α, $1\leq\alpha\leq y$, and β, $1\leq\beta\leq z$, we have the following constraint:

$$n_i^{\alpha,\beta}\geq\sum_{1\leq\ell\leq j!} m_{i,\ell}^{\alpha,\beta}.$$

And also we must restrict the total cost of all our bribes to the bribing limit k. Thus we need the following constraint:

$$k\geq\sum_{1\leq\alpha\leq y}\sum_{1\leq\beta\leq z}\sum_{1\leq i\leq j!}\sum_{1\leq\ell\leq j!} p_\beta m_{i,\ell}^{\alpha,\beta}.$$

These additional constraints will enforce a legal selection of voters to bribe. These, together with our specification of $D(a,b)$, complete the specification of the WCF-enforcing ILPFP. Overall we have an structure that will have a constant number of constraints in terms of the parameters, and that will have a solution if and only if there is a successful bribery. Also there will only be a constant number of such objects for every fixed set of parameter values (in fact it will be constant in terms of just the number of candidates, and there will be no more WCFs than in the unweighted unpriced case).

Thus the weighted, priced, and weighted and priced variants of bribery are all in FPT when parameterized on the number of candidates as well as on the cardinalities of the weight and/or price sets, as present. ❑

Pushing beyond this, for the case of manipulation (still parameterized by the number of candidates), we can handle even the case of there not being any bound on the cardinality of the weight set of all the voters, but rather there simply being a bound on the cardinality of the set of weights over all *manipulative* voters.

Theorem 7.2 *For Schulze elections and for (with any feasible tie-breaking functions) ranked pairs, coalitional weighted manipulation is in FPT with respect to the combined parameter "number of candidates" and "cardinality of the manipulators' weight set" in both the succinct and nonsuccinct input models, for both constructive and destructive manipulation, and in both the nonunique-winner model and the unique-winner model.*

Proof. Again let the candidates be $1, \ldots, j$ with candidate 1 being the distinguished candidate. The input specifies the nonmanipulator votes as a set of votes along with their weights, and it specifies the set of manipulator votes as a vector of pairs $((w_1, t_1), \ldots, (w_s, t_s))$ with there being t_i manipulators having weight w_i. Our top level loop will be essentially as in the unweighted manipulation case in Section 5.2.

Now, to handle this, we must be careful; since there is no limit on the overall number of weights, we can't have variables capturing how many voters of each occurring voter weight there are before manipulation. Rather, we in our looping algorithm that is building the ILPFPs use the power of our algorithm to itself precompute all the parts of the sums (appearing in the constraints) regarding all the nonmanipulators—so it is our looping algorithm that is putting in place constants that express the sums of the weights of nonmanipulative voters that have various properties: Namely, we have a constant $n_{a,b}$ describing the total weight of the nonmanipulative voters casting votes that prefer candidate a to candidate b for every pair of candidates a, b. The values $((w_1, t_1), \ldots, (w_s, t_s))$ describing the weights and how many manipulators there are of each weight will be constants of the ILPFP as well. The variables of the ILPFP describe how many manipulators of each weight get manipulated to each particular vote type: m_i^α for every i, $1 \leq i \leq j!$ and α, $1 \leq \alpha \leq s$, describes how many manipulators of weight w_α are assigned to vote type i.

The WCF can be implemented in terms of *Bigger* and *StrictlyBigger* predicates as before, with these implemented in terms of $D(a, b)$. In this case we will express $D(a, b)$ with the following:

$$n_{a,b} + \sum_{1 \leq \alpha \leq s} \sum_{i \in \mathit{pref}(a,b)} (w_\alpha m_i^\alpha) - n_{b,a} - \sum_{1 \leq \alpha \leq s} \sum_{i \in \mathit{pref}(b,a)} (w_\alpha m_i^\alpha).$$

Besides this we just need a few extra constraints to ensure that the manipulation chosen is a reasonable one. We include a constraint $m_i^\alpha \geq 0$ for every i, $1 \leq i \leq j!$, and α, $1 \leq \alpha \leq s$, enforcing that we do not include a negative number of voters of any type. And we include a constraint $\sum_{1 \leq i \leq j!} m_i^\alpha = t_\alpha$ for every α, $1 \leq \alpha \leq s$, ensuring that we use the same number of manipulators of every weight as we have available.

With these additions, we can build an ILPFP to encode our WCF that will be polynomially bounded in size with fixed values for the two parameters, and there will only be a constant number of such WCFs with fixed parameter values. Thus our algorithm specified above will run in uniformly polynomial time with fixed parameter values and this problem is in FPT. ❑

This still is all a valid framework for our many-uses-of-Lenstra-based approach. Note that for the case of the cardinality of the set of weights being 1, that gives the case of weighted noncoalitional manipulation mentioned as an aside by Dorn and Schlotter [DS12], though here we're handling even any fixed-constant number of manipulators (since any fixed-constant number have at most a fixed-constant cardinality of their weight set), and indeed, even a number of manipulators whose cardinality isn't bounded but who among them in total have a fixed-constant cardinality of occurring weights.

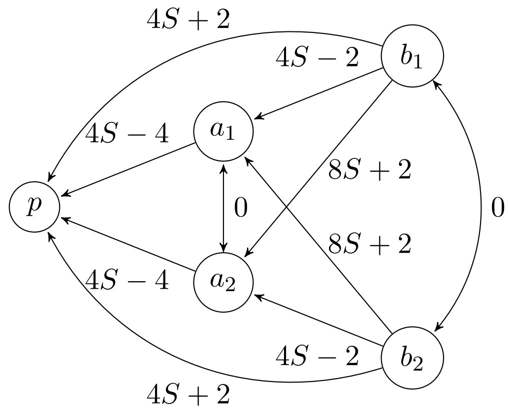

Figure 3: The important part of the WMG with just the nonmanipulators having cast their votes.

It seems intuitively necessary to bound the number of cardinality of the manipulator weight set to achieve results like the above. For ranked pairs we show that that is necessary: We prove that WCM (with an unbounded number of manipulator weights) is NP-complete in ranked pairs for any fixed number of candidates starting at five. Therefore there can't be any algorithm that is polynomial for any fixed number of candidates at least five unless P = NP. As a weaker consequence, this will also block the existence of an FPT algorithm for this problem parameterized solely on the number of candidates.

Theorem 7.3 *For every fixed number of candidates $j \geq 5$, weighted coalitional manipulation is NP-complete for ranked pairs.*

Proof. Containment in NP is trivial. To prove NP-hardness we will reduce from the NP-hard problem partition.

Definition 7.4 (Partition) *Given a set of n integers $A = \{a_1, \ldots, a_n\}$, does there exist a partition of A into two sets A_1 and A_2 such that $\sum_{x \in A_1} x = \sum_{y \in A_2} y$?*

Given a partition instance A, we will construct a weighted manipulation instance (C, V, W, p), where C is the candidate set, V is the set of nonmanipulative votes along with their weights, W is the set of manipulator weights, and p is the distinguished candidate.

The candidate set C will contain five important candidates: the distinguished candidate p and four other candidates: a_1, a_2, b_1, and b_2. There will also be $j - 5$ extra candidates which we will ensure are always easily beaten by p.

Let $S = \sum_{i=1}^{n} a_i$. The nonmanipulators will be assigned votes and weights such that the important candidates induce the WMG shown in Figure 3, and for every extra candidate e and every important candidate $c \in \{a_1, a_2, b_1, b_2, p\}$, $D(c, e) = 8S + 2$ (so that these edges remain strong regardless of the manipulator votes).

We enforce that our tie-breaking order favors both of the pairs $\{a_1, b_1\}$ and $\{a_2, b_2\}$ over the pairs $\{b_2, p\}$ and $\{b_1, p\}$, with the rest of the order being irrelevant.

The manipulator weight set W will be $\{4a_1, 4a_2, \ldots, 4a_n\}$.

Now we will show that there is a solution for the partition instance if and only if there is a solution for the manipulation instance.

($\leftarrow$) Suppose there is a solution for the manipulation instance. Namely, there is a set V' of n votes with the weights W such that p wins the election $(C, V \cup V')$. Let us use $D'(a, b)$ to denote the weight of the WMG edge (a, b) after manipulation. Note that the manipulators can always rank p at the top position without hurting p's chance of winning. Therefore, we will have $D'(p, a_1) = D'(p, a_2) = 4$ and $D'(b_1, p) = D'(b_2, p) = 2$. Also the weight of the edges from p to the extra candidates will become so strong that their respective order with p is fixed before the orders between p and the important candidates. So, in the final ranking, p is clearly placed higher than those candidates (we do not care about the relative order between pairs of extra candidates, since that does not affect p's performance). Also, we will have $D'(b_1, a_2) \geq 4S + 2$ and $D'(b_2, a_1) \geq 4S + 2$, regardless of the manipulator votes. These edges will then be considered before every other edge in the subgraph induced by the important candidates. Therefore, the orders $b_1 \succ a_2$ and $b_2 \succ a_1$ will be fixed in the final ranking. We argue that either $D'(a_1, b_1) = 2$ or $D'(a_2, b_2) = 2$. Otherwise there won't be any path of positive weight edges from p to b_1 and b_2, and we will eventually have $b_1 \succ p$ and $b_2 \succ p$ in the final ranking.

Now, without loss of generality, assume that $D'(a_1, b_1) = 2$. We argue that $D'(b_1, b_2) = 0$. Otherwise, we would have one of two cases (note that the manipulator weights are all multiples of 4, and $D(b_1, b_2) = 0$, so $D'(b_1, b_2)$ must be a multiple of 4):

1. $D'(b_1, b_2) \geq 4$: In this case, (b_1, b_2) will be considered before (a_1, b_1) and the order $b_1 \succ b_2$ will be fixed in the final ranking. Since we already have the order $b_2 \succ a_1$, by transitivity, we have $b_1 \succ a_1$ and (a_1, b_1) gets discarded. Therefore, the order between p and b_1 will not be specified by any transitive fixed orders until when the edge (b_1, p) is considered. Then p will not be a winner.

2. $D'(b_2, b_1) \geq 4$: We have two subcases here.

   (a) $D'(a_2, b_2) = 2$: In this case, (b_2, b_1) will be considered before (a_2, b_2) and the order $b_2 \succ a_2$ will be fixed by transitivity before considering that edge. Then the only positive incoming edge to b_2, which is (a_2, b_2), gets discarded, and eventually, when (b_2, p) is considered, p will be ranked lower than b_2.

   (b) $D'(b_2, a_2) \geq 2$: Since there would not be any positive incoming edge to b_2, p will be ranked lower than b_2 when considering (b_2, p).

Therefore, it must hold that $D(b_1, b_2) = 0$. Now we can build a solution for the partition problem as follows. For every manipulator i (which has the weight $4a_i$) having ranked b_1 higher than b_2, we make the partition A_1 include a_i, and for every manipulator j (which has the weight $4a_j$) having ranked b_2 higher than b_1, we make the partition A_2 include a_j. Then (A_1, A_2) is obviously a solution for the partition problem.

($\rightarrow$) Now, suppose there is a solution for the partition problem. That is, we have a partition of $\{a_1, \ldots, a_n\}$ into A_1 and A_2, such that $\sum_{x \in A_1} x = \sum_{y \in A_2} y$. We make every manipulator i with $a_i \in A_1$ cast the following vote:

$$p > a_1 > a_2 > b_1 > b_2 > \ldots$$

(where $[\ldots]$ denotes the extra candidates in any arbitrary order), and make every manipulator j with $a_j \in A_2$ cast the following vote:

$$p > a_2 > a_1 > b_2 > b_1 > \ldots .$$

After the manipulation, we will have $D'(a_1, a_2) = D'(b_1, b_2) = 0$ and $D'(a_1, b_1) = D'(a_2, b_2) = 2$. So, after having fixed in the final ranking the orders $b_1 \succ a_2$, $b_2 \succ a_1$, $p \succ a_1$, and $p \succ a_2$, the edges (a_1, b_1) and (a_2, b_2) will be considered, giving the final orders $p \succ a_1 \succ b_1$ and $p \succ a_2 \succ b_2$. So, we will have a transitive order from p to both b_1 and b_2 and p will be a winner. ❑

References

[BBCN12] N. Betzler, R. Bredereck, J. Chen, and R. Niedermeier. Studies in computational aspects of voting—A parameterized complexity perspective. In *The Multivariate Algorithmic Revolution and Beyond*, pages 318–363. Springer-Verlag *Lecture Notes in Computer Science #7370*, 2012.

[Bet10] N. Betzler. *A Multivariate Complexity Analysis of Voting Problems*. PhD thesis, Friedrich-Schiller-Universität Jena, Jena, Germany, 2010.

[BTT89] J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[BU09] N. Betzler and J. Uhlmann. Parameterized complexity of candidate control in elections and related digraph problems. *Theoretical Computer Science*, 410(52):43–53, 2009.

[CSL07] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):Article 14, 2007.

[DS12] B. Dorn and I. Schlotter. Multivariate complexity analysis of swap bribery. *Algorithmica*, 64(1):126–151, 2012.

[FHH09] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence*, 35:485–532, 2009.

[FHH11] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. *Journal of Artificial Intelligence*, 40:305–351, 2011.

[FHHR09] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence*, 35:275–341, 2009.

[Len83] H. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[LZ10] H. Liu and D. Zhu. Parameterized complexity of control problems in maximin election. *Information Processing Letters*, 110(10):383–388, 2010.

[McG53] D. McGarvey. A theorem on the construction of voting paradoxes. *Econometrica*, 21(4):608–610, 1953.

[MS12] C. Menton and P. Singh. Manipulation and control complexity of Schulze voting. Technical Report arXiv:1206.2111v3 [cs.GT], arXiv.org, September 2012.

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[PX12] D. Parkes and L. Xia. A complexity-of-strategic-behavior comparison between Schulze's rule and ranked pairs. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1429–1435. AAAI Press, August 2012.

[Sch11] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36:267–303, 2011.

[Tid06] N. Tideman. *Collective Decisions and Voting: The Potential for Public Choice*. Ashgate Publishing, 2006.

[Wik12] Wikipedia. Schulze method. en.wikipedia.org/wiki/Schulze_method, 2012.

[Xia12] L. Xia. Computing the margin of victory for various voting rules. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 982–999, June 2012.

[XZP+09] L. Xia, M. Zuckerman, A. Procaccia, V. Conitzer, and J. Rosenschein. Complexity of unweighted manipulation under some common voting rules. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 348–353, July 2009.